IE 498 - HW1

> Julius Olson

Implementation

Instructions & requirements

```
# Train
python3 nn.py train


# Load and evaluate
python3 nn.py load



# More advanced usage to customize settings

usage: nn.py [-h] [--hidden HIDDEN] [--epochs EPOCHS] [--data DATA]
             [--model MODEL]
             {train,load}

Neural Network

positional arguments:
  {train,load}     Run NN training or load exisiting model

optional arguments:
  -h, --help       show this help message and exit
  --hidden HIDDEN  Number of hidden layers
  --epochs EPOCHS  Number of epochs
  --data DATA      Dataset destination
  --model MODEL    .npy model destination
```

Requires python ver `3.6` or higher (due to usage of f-string formatting) and numpy.

Imlpementation

The main part of the program is the class `NeuralNetwork`, which contains all relevant methods needed for training and evaluating the network. Other than that, auxillary functions include different activation functions and a function for loading the data set from file.

Both the forward and backward algorithm utilize vectorization via numpy for faster calculations. The training is carried out using stochastic gradient descent.

Forward

```
"""
                Forward propagate
        """
        def forward(self, x):
                self.Z = np.dot(self.W, x) + self.b1
                self.H = sigmoid(self.Z)
                self.U = np.dot(self.C, self.H) + self.b2
                return softmax(self.U)
```

Backward

```
def backpropagate(self, x, y, out):
        dPdU = out
        dPdU[y]  -= 1
        dPdB2 = dPdU
        dPdC  = np.dot(dPdU, self.H.T)
        Sigma = np.dot(self.C.T, dPdU)
        dPdB1 = Sigma * back_sigmoid(self.Z)
        dPdW  = np.dot(dPdB1, x.T)
        return dPdW, dPdB1, dPdB2, dPdC, Sigma
```

Result

An accuracy of 97.4% was reached using the following settings.

Param	Value
`d_hidden`	100
`epochs`	10
`LR`	Piecewise constant (see code)
Activation	`sigmoid`